UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/698,509	10/31/2003	Dhruva Ranjan Chakrabarti	200314557-1	9606

22879 7590 09/17/2009

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
3404 E. Harmony Road
Mail Stop 35
FORT COLLINS, CO 80528

EXAMINER	
DAO, THUY CHAN	
ART UNIT	**PAPER NUMBER**
2192	

NOTIFICATION DATE	DELIVERY MODE
09/17/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

JERRY.SHORMA@HP.COM
ipa.mail@hp.com
jessica.l.fusek@hp.com

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/698,509
Filing Date: October 31, 2003
Appellant(s): CHAKRABARTI ET AL.

Dan C. Hu
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed June 8, 2009 appealing from the Office action mailed January 7, 2009.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

7,080,366	Kramskoy et al.	07-2006
6,651,243	Berry et al.	11-2003

(9) Grounds of Rejection

The following grounds of rejection are applicable to the appealed claims:

- ❑ Claims 1-13 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kramskoy (US Patent No. 7,080,366) in view of Berry (US Patent No. 6,651,243).

Claim 1:

Kramskoy discloses *a method of compiling a computer program with inline specialization, the method comprising:*

given a call stack/call frames (e.g., FIG. A5, col.63: 8 – col.64: 42; col.39: 14-23),

if multiple call-chains (call-chains 1068 → 1070 → 1072 → 1078; 1068 → 1070 → 1072 → 1080; 1072 → 1080 → 1083; or 1072 → 1080 → 1082, …) *… have a common call site* (e.g., FIG. 1E, col.20: 10-49, <u>1072 is a common call site</u> of 1078 and 1080; <u>1080 is a common call site</u> of 1082 and 1083; and/or <u>1084 is a common call site</u> of 1085 and 1086, emphasis added),

inlining the common call site in one or more of the call-chains (e.g., FIG. 1E, inlining the common call site 1072 in the call-chain 1068 → 1070 → 1072 → 1080 but not in the other call-chain 1068 → 1070 → 1072 → 1078; or inlining the common call site 1080 in the call-chain 1072 → 1080 → 1083, but not in the other call-chain 1072 → 1080 → 1082),

> "Where <u>the fragments of code are compiled</u>, preferably the compiler is able to optimise the compiled code. <u>Such optimisations might include inlining</u>…" (col.6: 47-49, emphasis added);
>
> "The compiler 1058 determines that the dominant path is therefore 1068, 1070, <u>1072, 1080</u>, 1083, 1084, 1085 through the code. The dominant path is indicated as 1088." (col. 20: 39-41, Code 1072 and Code 1080 (common call sites) are inlined in the call chain "1068, 1070, 1072, 1080, 1083", …, emphasis added); and

> "<u>Compiling dominant paths</u> of execution allows loop optimisations and <u>inlining</u> to be performed, while simplifying the analysis required for many optimizations…" (col.11: 11-13, emphasis added)

without inlining the common call site into all of said multiple call-chains having the common call site (e.g., FIG. 1E, without inlining 1078, 1082, 1086 into said dominant path, col.20: 31-62).

Kramskoy does not explicitly disclose a *call-graph.*

However, in an analogous art, Berry further discloses *a call-graph* (e.g., FIG. 11A-B and related text).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry's teaching into Kramskoy's teaching. One would have been motivated to do so as one of many ways to reflects the call stacks as suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table).

Claim 2:

The rejection of claim 1 is incorporated. Kramskoy discloses *whenever a call site from routine x to routine y is inlined, new call sites are added from routine x to all routines inlinable within routine y* (e.g., col.79: 6-36; col.85: 48-67).

Claim 3:

The rejection of claim 2 is incorporated. Berry further discloses *materialization of summary information for the new call sites added to the call-graph* (e.g., col.18: 19-49, FIG. 11-12 and related text).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry's teaching into Kramskoy's teaching. One would have been motivated to do so as one of many ways to reflects the call stacks as suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table).

Claim 4:

The rejection of claim 3 is incorporated. Kramskoy discloses *addition of the new call sites to a global work-list so that the new call sites are considered for inlining* (e.g., col.85: 48-67; col.86: 36-61).

Claim 5:

The rejection of claim 4 is incorporated. Kramskoy discloses *addition of dependence relationships between call sites, wherein if a new call site, y, is added because of inlining of call site, x, then y is dependent on x* (e.g., col.86: 36-61; col.84: 13-64).

Claim 6:

The rejection of claim 5 is incorporated. Kramskoy discloses *patching of the new call site, y, during inline transformation of call site, x, and generating an intermediate transformation for the new call site, y* (e.g., col.79: 6-36; col.86: 6-35).

Claim 7:

Kramskoy discloses *an apparatus for compiling a computer program with inline specialization* (e.g., FIG. A5, col.63: 8 – col.64: 42; col.39: 14-23), *the apparatus comprising:*

memory configured to store computer-readable instructions and data; a processor configured to access said memory and to execute said computer-readable instructions (e.g., FIG. 1E, col.20: 10-49, common call sites as 1072, 1080, 1084),*;*

computer-readable instructions stored in said memory and configured to inline a common call site in one or more call-chains in a call stack/call frames (e.g., FIG. 1E, inlining the common call sites 1072, 1080, 1084 in a dominant path 1088, col.20: 27 – col.21: 20; col.39: 14-23; col.13: 44-49; col.39: 10-38),

without inlining the common call site into all call-chains having the common call site (e.g., FIG. 1E, without inlining 1078, 1082, 1086 into said dominant path, col.20: 31-62).

Kramskoy does not explicitly disclose a *call-graph.*

However, in an analogous art, Berry further discloses *a call-graph* (e.g., FIG. 11A-B and related text).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry's teaching into Kramskoy's teaching. One would have been motivated to do so as one of many ways to reflects the call stacks as suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table).

Claim 8:

The rejection of claim 7 is incorporated. Kramskoy discloses *whenever a call site from routine x to routine y is inlined, new call sites are added from routine x to all routines inlinable within routine y* (e.g., col.84: 13-64; col.86: 6-35).

Claim 9:

The rejection of claim 8 is incorporated. Berry further discloses *materialization of summary information for the new call sites added to the call- graph is performed* (e.g., col.18: 19-49, FIG. 11-12 and related text).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry's teaching into Kramskoy's teaching. One would have been motivated to do so as one of many ways to reflects the call stacks as suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table).

Claim 10:

The rejection of claim 9 is incorporated. Kramskoy discloses *the new call sites are added to a global work-list so that the new call sites are considered for inlining* (e.g., col.85: 48-67; col.86: 6: 35).

Claim 11:

The rejection of claim 10 is incorporated. Kramskoy discloses *dependence relationships are created between call sites* (e.g., col.86: 6-35; col.79: 6-36).

Claim 12:

The rejection of claim 11 is incorporated. Kramskoy discloses *the inline transformation patches up an intermediate representation of the new call sites (by considering the dependence relationships) before potentially inlining the new call sites* (e.g., col.86: 36-61; col.84: 13-64).

Claim 13:

Kramskoy discloses *a computer-readable storage medium storing a computer program in executable form,*

the computer program being a source code compiler with cross-module optimization (e.g., FIG. A5, col.63: 8 – col.64: 42; col.39: 14-23),

the compiler including an inline specialization feature such that given a call stack/call frames (e.g., FIG. 1E, col.20: 10-49, common call sites as 1072, 1080, 1084),

if multiple call-chains in the call stack/call frames have a common call site (e.g., FIG. 1E, inlining the common call sites 1072, 1080, 1084 in a dominant path 1088, col.20: 27 – col.21: 20; col.39: 14-23; col.13: 44-49; col.39: 10-38),

the common call site is inlined in one or more of the call-chains, without having to inline the common call site into all of the multiple call-chains having the common call site (e.g., FIG. 1E, without inlining 1078, 1082, 1086 into said dominant path, col.20: 31-62).

Kramskoy does not explicitly disclose a *call-graph.*

However, in an analogous art, Berry further discloses *a call-graph* (e.g., FIG. 11A-B and related text).

It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry‘s teaching into Kramskoy‘s teaching. One would have been motivated to do so as one of many ways to reflects the call stacks as suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table).

-o-o-o-

(10) Response to Argument

A. Claims 1-13 were rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent No. 7,080,366 (Kramskoy) in view of U.S. Patent No. 6,651,243 (Berry).

1. Claims 1, 7, 13.

Limitations at issue "*if multiple call-chains ... have a common call site, inlining the common call site in one or more of the call-chains*" (e.g., claim 1, lines 3-4 and Brief, page 6, last paragraph - page 7).

Examiner respectfully disagrees with Appellants' assertions. Here, Kramskoy teaches such "call-chains" and "common call sites" as illustrated in FIG. 1E with annotation of "common call site" as - -O- - added below:

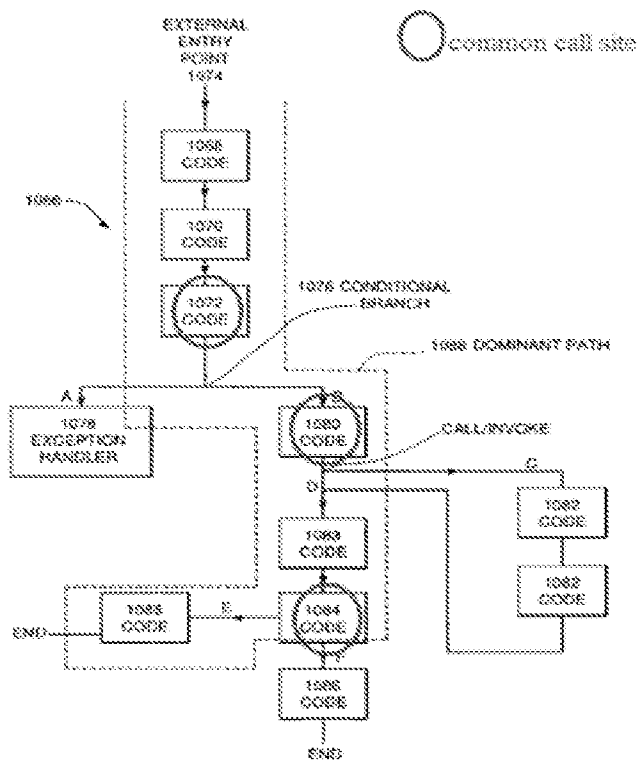

FIG. 1E

entry point 1074 → 1068 → … → 1084 → end point 1085 as a sequence of method calls, i.e., call-chains (e.g., col.13: 44-46, "When a compilation triggers, the dominant path (call-chain) can be determined by following the most popular successors a block at a time (relationship between blocks/functions/routines), including following <u>method calls</u>" - <u>method calls as routine calls</u>, emphasis added);

Code 1072 calls either 1078 or 1080 (e.g., col.20: 15-20, emphasis added); and
Code 1080 calls either 1082 or 1083 (e.g., col.20: 19-23, emphasis added).

It should be noted that in Kramskoy, "method calls" should be apparent to one skilled in the art as routine calls, thus, such a dominant part 1088 is a dominant call-chain, wherein the dominant path/call-chain has Code 1072 (common call site of 1078 and 1080) and Code 1080 (common call site of 1082 and 1083) inlined within – *See FIG. 1E, execution paths/call-chains 1072-path A-1078, 1072-path B-1080, 1080-path C-1082, 1080-parth D-1083…)*.

Kramskoy explicitly teaches:

if multiple call-chains (call-chains 1068 → 1070 → 1072 → 1078; 1068 → 1070 → 1072 → 1080; 1072 → 1080 → 1083; or 1072 → 1080 → 1082, …) … *have a common call site* (e.g., FIG. 1E, col.20: 10-49, 1072 is a common call site of 1078 and 1080; 1080 is a common call site of 1082 and 1083; and/or 1084 is a common call site of 1085 and 1086, emphasis added),

inlining the common call site in one or more of the call-chains (e.g., FIG. 1E, inlining the common call site 1072 in the call-chain 1068 → 1070 → 1072 → 1080 but not in the other call-chain 1068 → 1070 → 1072 → 1078; or inlining the common call site 1080 in the call-chain 1072 → 1080 → 1083, but not in the other call-chain 1072 → 1080 → 1082),

> "Where the fragments of code are compiled, preferably the compiler is able to optimise the compiled code. Such optimisations might include inlining…" (col.6: 47-49, emphasis added);
>
> "The compiler 1058 determines that the dominant path is therefore 1068, 1070, 1072, 1080, 1083, 1084, 1085 through the code. The dominant path is indicated as 1088." (col. 20: 39-41, Code 1072 and Code 1080 (common call sites) are inlined in the call chain "1068, 1070, 1072, 1080, 1083", …, emphasis added); and

> "Compiling dominant paths of execution allows loop optimisations and inlining to be performed, while simplifying the analysis required for many optimizations…" (col.11: 11-13, emphasis added).

Appellants further asserted,

> "The secondary, reference, Berry, cited by the Examiner also provides no teaching or hint of the inlining that is performed according to claim 1, where a common call site in a call-graph is inlined in one or more call-chains without inlining the common call site into all of the multiple call-chains having the common call site. Berry shows a call stack tree. Assuming for the sake of argument that a call stack tree can constitute a call-graph (which it does not), it is respectfully submitted that there is absolutely no hint given in Berry of inlining a common call site in such a call stack tree in one or more call-chains, without inlining the common call site into all of the multiple call-chains having the common call site." (Brief, page 7, emphasis added).

Examiner notes that Berry is not relied upon to reject particular limitations which Appellants argued about. As set forth in the previous Office action mailed January 7, 2009 (page 6), where Berry is applied to show such a known "call-graph" in the art as follows:

> "Kramskoy does not explicitly disclose a *call-graph.*
>
> "However, in an analogous art, Berry further discloses a call-graph (e.g., FIG. 11A-B and related text).
>
> It would have been obvious to a person having ordinary skill in the art at the time the invention was made to combine Berry's teaching into Kramskoy's teaching. One would have been motivated to do so as one of many ways to reflect the call stacks as

> suggested by Berry (e.g., col.18: 19-49, FIG. 11-12, call stacks presented as either tree or table)".

Appellants further asserted,

> "Moreover, it is respectfully submitted that neither Kramskoy nor Berry discloses a call-graph as recited in claim 1. As understood by persons of ordinary skill in the art, <u>a call-graph is a graph that indicates relationships between routines of a program</u>…" (Brief, page 7, last paragraph to page 8, first line, emphasis added).

As an initial matter, examiner notes that Appellants acknowledged that <u>call stacks disclosed by Kramskoy and Berry</u> store information of actively <u>executing routines of a program</u>:

> "In contrast, the call stacks disclosed by Kramskoy and <u>Berry</u> are data structures that store information about actively <u>executing routines of a program</u> …" (Brief, page 8, first paragraph, lines 1-2, emphasis added).

Kramskoy further explicitly teaches said stored information in Kramskoy's call stack indicating "*relationships between routines of a program*" - col.20: 10-38 with emphasis added as below:

> FIG. 1E shows paths of execution through code of a method generally referred to as **1066**.
>
> The figure shows schematically various fragments of code, for example **1068**, **1070**, **1072**. Such fragments of code may each represent a block of code.
>
> The code shown in the Figure has one external entry point **1074**. After block **1072**, there is a conditional branch **1076**, for example an exception check. If an exception occurs, the execution passes along path A to code **1078** to handle the exception. Otherwise, code passes along path B to code block **1080** at which point there may be a call (path C to block **1082**) or the execution may follow path D to code sections **1083**, **1084**. Execution may pass along path E to block **1085** or path F to block **1086**.
>
> Information about execution runs through the code **1066** is recorded on the execution history recorder **1050** run by the interpreter **1042**.
>
> If block **1068** is found to have been executed by the interpreter the threshold number of times, it is passed to the queue **1054**. The compiler **1058** consults the execution history in the recorder **1050** and finds that:
>
> 1. The more popular successor of **1072** is **1080** (that is, execution passed along path B more often than along path A);
> 2. The more popular successor of **1080** is **1083** (that is, execution passed along D more often than along C); and
> 3. The more popular successor of **1084** is **1085** (that is, execution passed along D more often than along C).

It should be noted that the relationships between method calls (col.13: 44-46), routine calls and/or fragments of code/blocks of code are at least indicated by the arrow execution flow directions – *See FIG. 1E, execution paths/call-chains 1072-path A-1078, 1072-path B-1080, 1080-path C-1082, 1080-parth D-1083…).*

Appellants further asserted,

> "In contrast, the call stacks disclosed by Kramskoy and Berry are data structures that store information about actively

> <u>executing routines of a program</u>. The Office Action cited Fig. A5 of Kramskoy as purportedly disclosing a call-graph. However, as specifically stated by Kramskoy, Fig. A5 depicts a call stack. <u>Similarly, the call stack tree of Berry</u> "reflects the call stacks recorded to date." Berry, 18:24." (Brief, page 8, emphasis added).

Examiner respectfully disagrees. As set forth in the previous Office action mailed January 7, 2009 (page 4) reproduced in part below, where Berry explicitly teaches call stacks are reflected/implemented as call tree/graph:

> "With reference now to <u>FIG. 11A</u>, a diagram depicts a tree structure generated from trace data. This figure illustrates a <u>call stack tree</u> 1100 in which <u>each node in tree structure 1100 represents a function entry point</u>" (col.15: 58-61, call stack implemented as a tree, emphasis added); and
>
> "With reference now to <u>FIG. 11B</u>, <u>a call stack tree</u> which reflects <u>call stacks</u> observed during a specific example of system execution will now be described. <u>At each node in the tree</u>, several statistics are recorded. In the example shown in FIG. 11B, the statistics are time-based statistics. The particular statistics shown include the number of distinct times the <u>call stack</u> is produced, the sum of the time spent in the call stack, the total time spent in the call stack plus the time in <u>those call stacks invoked from this call stack</u> (referred to as cumulative time), and the number of instances of this routine above this instance (indicating depth of recursion)" (col.16: 22-33, <u>call stacks are reflected/implemented as a call tree/graph</u>, emphasis added).

Regarding Appellants' arguments "*Moreover, it is respectfully submitted that <u>neither Kramskoy nor Berry discloses a call- graph</u> as recited in claim 1. As understood*

by persons of ordinary skill in the art, a call-graph is a graph that indicates relationships between routines of a program" (Brief, page 7, last paragraph to page 8, first line, emphasis added); and examiner also notes that in the Remarks filed October 13, 2008, first paragraph, Appellants characterized Berry's teachings:

> "*…Berry is cited in relation to FIGS. 11 and 12 as disclosing a call graph. However, applicants respectfully submit that FIGS. 11 and 12 of Berry show call stacks, not call graphs. Call stacks, whether in tree or table form, are not call graphs. As discussed above, a call graph indicates calling relationships.*" (emphasis added and reproduced in part herein).

The examiner respectfully disagrees with how Appellants characterized prior art teachings and would like to direct Appellants' attention also to page 4 of the previous Office action mailed January 7, 2009, reproduced in part below, in which the Appellants have not responded in the Brief:

> "…As well-known in the art, call trees are also called call graphs:
>
> **US Patent No. 6,539,543**, col.6: 36-41, "*Function call graph 20 is a call tree representing function calls in source code 10. It is used in the "flattening" phase and caches information for all function calls in the source code. Every function in the source code is represented by a node in the tree and every function call is represented by an edge*" (emphasis added);
>
> **US Patent No. 7,058,928**, col.2: 45-49, "*For example, the function call graph in a serial environment is a simple tree. In a parallel processing environment, the function call graph is no longer a simple tree, but a collection of trees*" (emphasis added); and
>
> **US Patent No. 6,026,362**, col.3: 64 – col.4: 5, "*For example, the debugger could display a tree-like graphical representation of the program, wherein each function comprising*

> *the program is represented by <u>a node on the graph and the call relationships</u> between functions are illustrated by <u>lines interconnecting the nodes</u>. <u>Such a graph is known as a call tree</u>*" (emphasis added)."

Thus, at least, what Appellants' so-called "a call-graph is a graph that indicate *indicates <u>relationships between routines of a program</u>"* (emphasis added) is well-known in the art as - -call tree- -, as particularly taught in US Patent 6,026,362 as noted above.

Kramskoy, FIG. 1E at least illustrated relationships between paths of execution (call-paths/chains) between method calls (col.13: 44-46), routine calls, and/or fragments of code/blocks of code – *See FIG. 1E, execution paths/call-chains 1072-path A-1078, 1072-path B-1080, 1080-path C-1082, 1080-parth D-1083…)*. That is to say "*relationships between routines of a program"* as Appellants considered and argued.

2. Claims 2, 3, 8, 9.

Limitations at issue, "…*whenever a call site from routine x to routine y is inlined, new call sites are added from routine x to all routines inlinable within routine y*" (Brief, page 9 regarding claim 2, lines 2-3).

Examiner respectfully disagrees with Appellants' assertions. Kramskoy explicitly teaches:

whenever a call site from routine x to routine y is inlined (e.g., FIG. A7-3, col. 85: 53-55, a call site from Second Section of Code (x) to "bar" (y), now "bar" (y) is inlined in said Second Section of Code),

new call sites are added from routine x to all routines inlinable within routine y (e.g., col.85: 49-53, 4032 calls 4034, which in turn calls inlined "bar" → 4034 is added from Second Section of Code (x) to all routines/instructions within inlined "bar" (y))

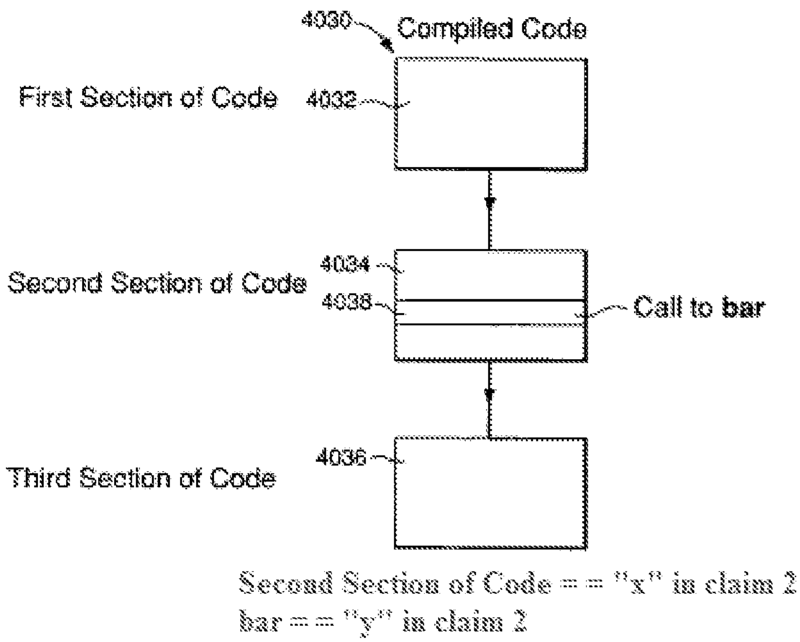

FIG. A7-3

3. Claims 4-6, 10-12.

Limitations at issue, "…*adding the new call sites to a global work-list so that the new call sites are considered for inlining*" (Brief, pp. 9-10 regarding claim 4, lines 2-3).

Examiner respectfully disagrees with Appellants' assertions. Kramskoy explicitly teaches *adding the new call sites to a global work-list so that the new call sites are considered for inlining* (e.g.,

> "In the following, the optimisation made in the compilation of the code with the assumption that <u>bar as final is in lining.</u> FIG. A7-3 shows sections of compiled code 4030 including a first section 4032, a second section 4034 and a third section 4036. The second section 4034 is a compiled version of code including a call

> to *bar* 4038. The Method bar has been in lined so that bar is now contained in the section 4034 as section 4038.
>
> If it is later found that the assumption is incorrect, compiled code section 4034 will be deleted. The dispatch table of the Method including the section 4034 is altered so as not to refer to the compiled version of the code 4034." (col.85: 48-60, dispatch table as "global work-list" to register the call sites, emphasis added).

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejection should be sustained.

Respectfully submitted,

/Twee Dao/

Examiner, Art Unit 2192

Conferees:

/Tuan Q. Dam/
Tuan Q. Dam
Supervisory Patent Examiner, Art Unit 2192

/Lewis A. Bullock, Jr./

Lewis A. Bullock, Jr.

Supervisory Patent Examiner, Art Unit 2193